UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/577,956	02/15/2007	Henrik Kjaer	ALB.024	5195

20987 7590 11/26/2010

VOLENTINE & WHITT PLLC
ONE FREEDOM SQUARE
11951 FREEDOM DRIVE SUITE 1260
RESTON, VA 20190

EXAMINER	
TRAN, QUOC A	
ART UNIT	PAPER NUMBER
2176	

NOTIFICATION DATE	DELIVERY MODE
11/26/2010	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

ptoinbox@volentine.com
cjohnson@volentine.com
aloomis@volentine.com

Office Action Summary	Application No.	Applicant(s)	
	10/577,956	KJAER, HENRIK	
	Examiner	Art Unit	
	QUOC A. TRAN	2176	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *07 September 2010*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-47* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-47* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *07 September 2010* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☒ Certified copies of the priority documents have been received in Application No. 10/577,956.

3.☒ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

This is a Final Office Action in responses to applicant's amendments and remarks filed 09/07/2010. The current patent application claims foreign priority to PA2003-01635 files ***11/03/2003*** and PA2004-01208 files 08/09/2004.

- Claims 21-47 are pending.
- Claims 1-20 were canceled.
- Claim 21 is independent claim.
- Claims 21, 24, 26, 32 and 39 have been amended
- Claims 41-47 are new.

In addition, regarding Claims 24 and 32, which were rejected under 112 second paragraph, is hereby withdrawn, in light of Applicant's amendments.

Claim Rejections - 35 USC § 112

The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claim(s) 41 and 44 rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim contains subject matter which was not described in the Specification in such a way as to reasonably convey to one skilled in the relevant art that the inventors, at the time the application was filed, had possession of the claimed invention. :

- Claim 41 recites the limitation "for *automatic generation of* a new panel" (see amended claim 41). There is **no** mention in the original/current Specification of this feature.
- Claim 44 recites the limitation "*automatic generation* of the required number of destination cells in the output panel" (see amended claim 44). There is **no** mention in the original/current Specification of this feature.

If the examiner has overlooked the portion of the original Specification that describes this feature of the present invention, then Applicant should point it out (by page number and line number) in the response to this Office Action.

Applicant may obviate these rejections by canceling the claims.

Claims Rejection – 35 U.S.C. 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 21-23, 25, 28-37 and 40-41, 44 and 46 are rejected under 35 U.S.C. 103(a) as being unpatentable over **Kotler** et al., (US 20090083615A1-Continuation of No. 10/961,313, which is a Division of No. 09/599,808 filed 06/21/2000 [hereinafter

"Kotler"], in view of **Spencer** et al., (US005603021A - filed 09/02/1994) [hereinafter "Spencer"].

Regarding ***independent claim 21***, Kotler teaches:

An electronic mathematical model builder

(In Kotler: at Para 43→ Kotler discloses this limitation, as clearly indicated in the cited text [e.g., the spreadsheet engine includes such functions as *formula creation,* (e.g., mathematical -see applicant current Specs @ page 6 lines 9-14).]

comprising a memory for storage of data, a processor for defining addressable panels of cells stored in the memory with a unique identifier, for entering data into the cells and for processing data stored in the cells,

(In Kotler: at Para 46→ Kotler describes Spreadsheet programs enable users to store the actual data for each cell in the table 124 or an associated free floating field 126, such as text, values, and formulas. A cell table 134 stores pointers [e.g., in the memory with a unique identifier -see applicant current Specs @ page 5 lines23-24) to one or more cells that physically store the data. A cell table associated with a table, such as cell table 134(1), contains multiple cells 136(1)-136(C), one for each cell in the table.)

a user interface with a display for displaying panels of cells in a work area and means for creating and individually and independently positioning panels of cells in the work area and means for specifying data to be entered into the cells,

(In Kotler: at Para 110 and illustrates in Fig. 3→ Kotler describes Spreadsheet programs enable users to perform a reference edit operation, in which the user references one or more cells to extract their data values for inclusion in a formula in another cell or free floating field. In FIG. 3, the user begins entering a summation formula (i.e., "=SUM") in the formula edit box 312 above cell C6. The user then references cells C2 through C5 using a pointer 310 or some other mechanism. The referenced cells C2:C5 are highlighted or otherwise indicated as being selected, as represented by the bold rectangular box around the cells.

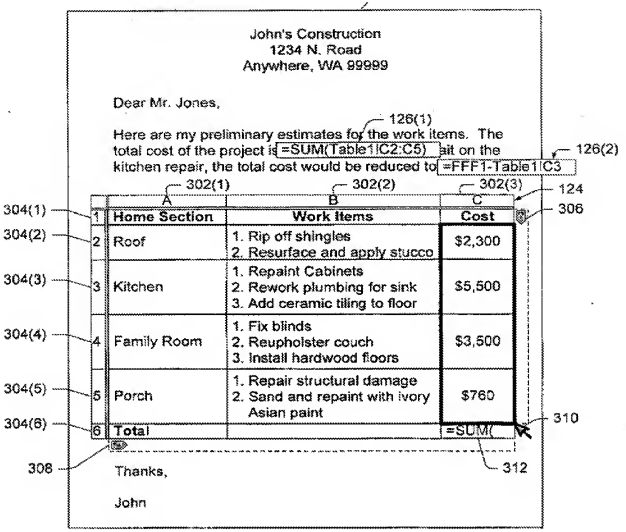

Fig. 3

In addition, Kotler further teaches the cross -table referencing, where table 606 contains references to tables 602 and 604, where a cell in one table or field contains a formula referencing a cell in another table. All three tables are SEAPATE AND INDEPENDNENT from one another, and architecturally have their own set of grid, spreadsheet and table objects 130, 106, and 104. In FIG. 6, cell B2 in table 606

contains a summation formula for adding values in cells B2 and B3 of table 602 (i.e., =SUM(Table1!B2:Table1!B3)). Cell B3 in table 606 contains a summation formula for adding values in cells B2 through B4 in table 604 (i.e., =SUM(Table2!B2:Table2!B4)) (In Kotler: @ Para(s) 114-115 and illustrates in Fig. 6).

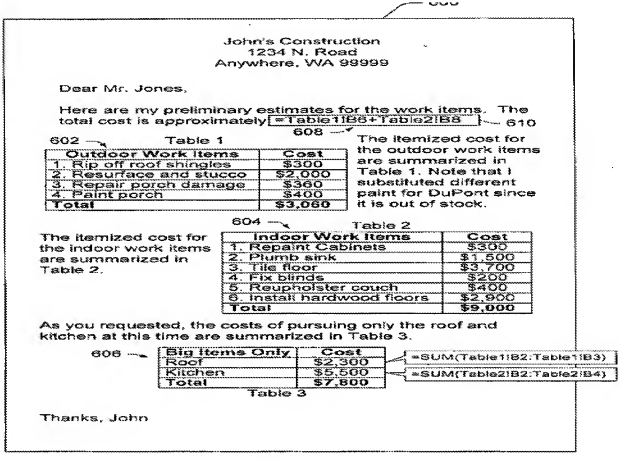

Fig. 6

Therefore, as broadly disclosed in the instant specification at Page 7, lines 5-6 and lines 8-9, which is states, 'In a preferred embodiment of the invention, *cells are created and displayed in sets or panels. ... A panel may hold one cell...*' , it is reasonable to find that the term, "*panels of cells* " is broad enough to reasonably interprets as "cells tables" in Kotler. Thus examiner concludes, reasonably, that the

claimed, displaying panels of cells in a work area and means for creating and individually and independently positioning panels of cells in the work area by Kotler.

and a function builder for establishing a function comprising mathematical relations between panels of cells, wherein the function builder is further adapted for generation of an output panel comprising a plurality of cells to hold output function values

(In Kotler: at Para 43→ Kotler discloses this limitation, as clearly indicated in the cited text [e.g., the spreadsheet engine includes such functions as *formula creation*. In addition Kotler further discloses the cross -table referencing, where table 606 contains references to tables 602 and 604, where a cell in one table or field contains a formula referencing a cell in another table. All three tables are SEAPATE AND INDEPENDNENT from one another, and architecturally have their own set of grid, spreadsheet and table objects 130, 106, and 104 (In Kotler: at Para(s) 114-116 and illustrates in Fig. 6).

In addition Kotler does not expressly teach, but Spencer teaches:

comprising fields for user specification of a desired function by mathematical operators,

(At Figure 3A and at the Abstract and Col. 11, Lines 10-15→ Spencer discloses a Formula Composer 136 is a visual tool for creating, editing, and debugging spreadsheet

formulas. Formula Composer Dialog providing multiple views of a formula, including: (1) a Formula Outline Pane to examine the structure of a formula, edit parts of the formula, and trace cell references and block names; (2) a Subexpression Edit Field for text editing of a formula or portion thereof; and (3) a Function Panel to facilitate input of ***@-functions and their arguments***. With these different views, the user can easily choose the right @-functions and enter all necessary information correctly. Also Spencer further teaches The "@" button 303, which appears to the left of the Subexpression Field, displays Functions Dialog 321, as shown in FIG. 3B. The Dialog 321 includes a Function Category List 323 and a Function (name) list 325. For example, that a user has selected the "Amortized Accumulated Interest" (AMAINT) function from the list 325. In response, the system updates the Dialog 321, as shown by the Dialog 321a in FIG. 3C. The AMAINT function is shown selected, at 325a. The information panel has been updated accordingly, shown at 327a. Upon the user selecting the OK button 329, the AMAINT @-function will be placed in the Subexpression Edit Field, as shown by the Dialog 301a of FIG. 3D. The user may now proceed to further change the formula using the Formula Outline Pane (In Spencer: @ Col. 12 Lines 10-35 and illustrates in Fig(s) 3(a), (b), (c) and (d)).

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler's spreadsheet formula model builder engine, to include a means of said comprising fields for user specification of a desired function by mathematical operators as taught by Spencer; because they are both from the

analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Spencer relates to general method of modeling spreadsheet formula in Kotler. This is done in an iterative manner that enabling the cross -table referencing, where a cell in one table or field contains a formula referencing a cell in another table (In Kotler: @ Para 114); also with this architecture, spreadsheet functionality is no longer constrained to grids, but is available in customary text and across table boundaries (In Kotler: @ Para 34). The ability to cross-reference other tables or free floating fields is beneficial in that all tables and free floating fields can be universally updated for any change in just one of the tables. For example, suppose the user changes the value in cell B2 of table 602 from $300 to $400. As a result of this change, table 602 is updated to reflect the new value $400 and the total amount in cell B6 is updated from $3,060 to $3,160. Additionally, the value in cell B2 of table 606 is updated to $2,400, causing the total amount in cell B4 in table 606 to be changed from $7,800 to $7,900 (In Kotler: @ Para 116 and see also Fig. 6).

Claims 22-23,

Kotler and Spencer teach the method of claim 21 and further comprise:

wherein the function builder comprises components for selection by the user for specification of the function; wherein at least one component is a function component selected from the GROUP CONSISTING OF

mathematical operators, mathematical functions, logical and Boolean functions, financial functions, string functions, and data base functions.

(In Kotler: at Para 110 and illustrates in Fig. 3→ Kotler describes Spreadsheet programs enable users to perform a reference edit operation, in which the user references one or more cells to extract their data values for inclusion in a formula in another cell or free floating field. In FIG. 3, the user begins entering a summation formula (i.e., "=SUM") in the formula edit box 312 above cell C6. The user then references cells C2 through C5 using a pointer 310 or some other mechanism. The referenced cells C2:C5 are highlighted or otherwise indicated as being selected, as represented by the bold rectangular box around the cells.)

Claim 25,

Kotler and Spencer teach the method of claim 22 and further comprise:

wherein at least one component is a data component including data of a set of cells.

(@ Para 110→ Kotler discloses this limitation, as clearly indicated in the cited text [e.g., Spreadsheet programs enable users to perform a reference edit operation, in which the user references one or more cells to extract their data values for inclusion in a formula in another cell or free floating field.])

Claim 28,

Kotler and Spencer teach the method of claim 21 and further comprise:

wherein the function builder comprises graphical symbols relating to respective components for selection by the user to be incorporated into the desired function.

(At Figure 3A and at the Abstract and Col. 11, Lines 10-15→ Spencer discloses a Formula Composer 136 is a visual tool for creating, editing, and debugging spreadsheet formulas. Formula Composer Dialog providing multiple views of a formula. Also Spencer further teaches The "@" button 303, which appears to the left of the Subexpression Field, displays Functions Dialog 321, as shown in FIG. 3B. The Dialog 321 includes a Function Category List 323 and a Function (name) list 325. For example, that a user has selected the "Amortized Accumulated Interest" (AMAINT) function from the list 325. In response, *the system updates the Dialog 321* (In Spencer: @ Col. 12 Lines 10-35 and illustrates in Fig(s) 3(a), (b), (c) and (d)).

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler's spreadsheet formula model builder engine, to include a means of said the function builder comprises graphical symbols relating to respective components for selection by the user to be incorporated into the desired function as taught by Spencer; because they are both from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Spencer relates to general method of modeling spreadsheet formula in Kotler. This is done in an iterative manner that enabling an electronic spreadsheet system includes a Formula Composer having a preferred

interface and methods for assisting a user with composing spreadsheet formulas. The Formula Composer also provides specific information about the selected spreadsheet function; the Formula Expert provides input fields which are specific for the arguments of the selected function. Moreover, the Formula Expert includes mode expressions or "templates" for assisting users in inputting correct argument information. Using pattern matching technique, the system may employ the templates for eliminating common user input mistakes. Methods are described for synchronizing the various views of the Formula Composer, so that a modification to the formula being edited by the user in one view is automatically and immediately propagated to the other views. In this fashion, all views remain synchronized during formula editing, thus allowing the user to easily switch among the views [In Spencer at the Abstract].

Claims 29-30,

Kotler and Spencer teach the method of claim 28 and further comprise:

wherein the graphical symbols relating to respective components are organized under various tabs according to their type; wherein the graphical symbols relating to respective components are organized in a graphical, hierarchical diagram according to their type.

(@ Col. 11 lines 10-25→ Spencer discloses Formula Composer 136 is a visual tool for creating, editing, and debugging spreadsheet formulas; wherein composer a function panel to facilitate input of @-functions and their arguments. With these different views, the user can easily choose the right @-functions and enter all necessary information

correctly. Also Spencer further discloses the graphical symbols relating to respective components are organized in a graphical, hierarchical diagram according to their type [@ Col. 23 lines 15-25].)

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler's spreadsheet formula model builder engine, to include a means of said the graphical symbols relating to respective components are organized under various tabs according to their type; wherein the graphical symbols relating to respective components are organized in a graphical, hierarchical diagram according to their type as taught by Spencer; because they are both from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Spencer relates to general method of modeling spreadsheet formula in Kotler. This is done in an iterative manner that enabling an electronic spreadsheet system includes a Formula Composer having a preferred interface and methods for assisting a user with composing spreadsheet formulas. The Formula Composer also provides specific information about the selected spreadsheet function; the Formula Expert provides input fields which are specific for the arguments of the selected function. Moreover, the Formula Expert includes mode expressions or "templates" for assisting users in inputting correct argument information. Using pattern matching technique, the system may employ the templates for eliminating common user input mistakes. Methods are described for synchronizing the various views of the Formula Composer, so that a modification to the formula being edited by the user in one

view is automatically and immediately propagated to the other views. In this fashion, all views remain synchronized during formula editing, thus allowing the user to easily switch among the views [In Spencer at the Abstract].

Claim 31,

Kotler and Spencer teach the method of claim 28 and further comprise:

wherein components are selected by dragging and dropping the corresponding graphical symbols into the function field of the function builder.

(In Kotler: Para 37→ Kotler discloses the UI manager 116 manages how the free floating fields and tables appear in a document and facilitates such tasks as table resizing, selection, cut, copy, paste, split, merge, table formatting and so on. The UI manager 116 includes the table object 104, the spreadsheet objects 106, and the spreadsheet editor 108)

Claim 32,

Kotler and Spencer teach the method of claim 28 and further comprise:

wherein the function builder comprises tools for specification of calculation type.

(In Kotler: Para 51→ Kotler discloses there are three types of formulas: normal, semi-calculation, and non-calculation. The normal formula is reevaluated only when its

dependencies change. The semi-calculation formula is reevaluated every time the recalculation engine 142 performs a recalculation operation. The non-calculation formula is never evaluated at all. Non-calculation formulas are a special formula type for handling nested tables (i.e., a table within a table) and nested free floating fields.)

Claim 33,

Kotler and Spencer teach the method of claim 28 and further comprise:

the user interface comprises means for naming functions built with the function builder;

(In Spencer: @ Col. 12 Lines 10-35 and illustrates in Fig(s) 3(a), (b), (c) and (d) → Spencer discloses the "@" button 303, which appears to the left of the Subexpression Field, displays Functions Dialog 321, as shown in FIG. 3B. The Dialog 321 includes a Function Category List 323 and a Function (name) list 325.)

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler's spreadsheet formula model builder engine, to include a means of said the user interface comprises means for naming functions built with the function builder as taught by Spencer; because they are both from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Spencer relates to general method of modeling spreadsheet formula in Kotler. This is done in an iterative manner that enabling an electronic spreadsheet system includes a Formula Composer having a

preferred interface and methods for assisting a user with composing spreadsheet formulas. The Formula Composer also provides specific information about the selected spreadsheet function; the Formula Expert provides input fields which are specific for the arguments of the selected function [In Spencer at the Abstract].

Claim 34,

Kotler and Spencer teach the method of claim 33 and further comprise:

and means for storage and retrieval of functions in the memory;

(In Kotler: Para 47→ Kotler discloses the formulas, such as the summation formula in cell C6 of FIG. 3, are stored as pointers to the appropriate formula object maintained by the formula manager 140.)

Claim 35,

Kotler and Spencer teach the method of claim 21 and further comprise:

and means for storing a selected part of the model in the memory;

(In Kotler: Para 47→ Kotler discloses the formulas, such as the summation formula in cell C6 of FIG. 3, are stored as pointers to the appropriate formula object maintained by the formula manager 140.)

.***Claim 36,***

Kotler and Spencer teach the method of claim 21 and further comprise:

wherein a first function may be an input variable to a second function;

(In Kotler: at Para 110 and illustrates in Fig. 3→ Kotler describes Spreadsheet programs enable users to perform a reference edit operation, in which the user references one or more cells to extract their data values for inclusion in a formula in another cell.)

Claim 37,

Kotler and Spencer teach the method of claim 21 and further comprise:

wherein the function builder further comprises tools for user definition of a mathematical operator.

(At Figure 3A and at the Abstract and Col. 11, Lines 10-15→ Spencer discloses a Formula Composer 136 is a visual tool for creating, editing, and debugging spreadsheet formulas. Formula Composer Dialog providing multiple views of a formula. Also Spencer further teaches The "@" button 303, which appears to the left of the Subexpression Field, displays Functions Dialog 321, as shown in FIG. 3B. The Dialog 321 includes a Function Category List 323 and a Function (name) list 325. For example, that a user has selected the "Amortized Accumulated Interest" (AMAINT) function from the list 325. In response, *the system updates the Dialog 321* (In Spencer: @ Col. 12 Lines 10-35 and illustrates in Fig(s) 3(a), (b), (c) and (d)).

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler's spreadsheet formula model builder engine, to

include a means of said the function builder further comprises tools for user definition of a mathematical operator as taught by Spencer; because they are both from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Spencer relates to general method of modeling spreadsheet formula in Kotler. This is done in an iterative manner that enabling an electronic spreadsheet system includes a Formula Composer having a preferred interface and methods for assisting a user with composing spreadsheet formulas. The Formula Composer also provides specific information about the selected spreadsheet function; the Formula Expert provides input fields which are specific for the arguments of the selected function. Moreover, the Formula Expert includes mode expressions or "templates" for assisting users in inputting correct argument information. Using pattern matching technique, the system may employ the templates for eliminating common user input mistakes. Methods are described for synchronizing the various views of the Formula Composer, so that a modification to the formula being edited by the user in one view is automatically and immediately propagated to the other views. In this fashion, all views remain synchronized during formula editing, thus allowing the user to easily switch among the views [In Spencer at the Abstract].

Claim 40,

Kotler and Spencer teach the method of claim 21 and further comprise:

further including tools for documenting the structure of a specific model document.

(In Kotler: Para 149→ Kotler discloses this limitation, as clearly indicated in the cited text [e.g., the Table object 104 manages and monitors the user input for structure changes, such as insertion/deletion of a row, merging cells, and so forth. When the user makes a structure change, the Table object 104 fires events to the GridBehavior object 150, which informs the spreadsheet engine 112, and in turn updates the cell table 134 associated with the UI table..])

Claim 41,

Kotler and Spencer teach the method of claim 21 and further comprise:

wherein the function builder is further adapted for automatic generation of a new panel of cells for containing values of the function.

(In Kotler: Para 139→ Kotler discloses when the user created the formula "=A1" in cell C1. The user selects cell C1, types in an "=" sign to indicate a formula, and then references the cell A1 by moving the mouse pointer 904 to cell A1 and clicking. The spreadsheet objects 106 (namely, CellEditing object 152) capture this reference and pass it to parser 144. The parser 144 recognizes it as a formula creates a formula object and inserts the formula into a cell of cell table 134, as indicated by cell C1. It is reasonable to interprets the steps of said the parser 144 recognizes it as a formula, creates a formula object and inserts the formula into a cell of cell table 134 "without user intervention",)

Therefore, as broadly disclosed in the instant specification at Page 7, lines 5-6 and lines 8-9, which is states, ´In a preferred embodiment of the invention, *cells are*

created and displayed in sets or panels. ... A panel may hold one cell...', it is reasonable to find that the term, "*panel of cells* " is broad enough to reasonably interprets as "cells table" in Kotler. Thus examiner concludes, reasonably, that the claimed, generation of a new panel of cells for containing values of the function is described by Kotler.)

Claim 44,

Kotler and Spencer teach the method of claim 21 and further comprise:

wherein the processor is further adapted for automatic generation of the required number of destination cells in the output panel.

(In Kotler: Para 139→ Kotler discloses when the user created the formula "=A1" in cell C1. The user selects cell C1, types in an "=" sign to indicate a formula, and then references the cell A1 by moving the mouse pointer 904 to cell A1 and clicking. The spreadsheet objects 106 (namely, CellEditing object 152) capture this reference and pass it to parser 144. The parser 144 recognizes it as a formula creates a formula object and inserts the formula into a cell of cell table 134, as indicated by cell C1. It is reasonable to interprets the steps of said the parser 144 recognizes it as a formula, creates a formula object and inserts the formula into a cell of cell table 134 "without user intervention",)

Therefore, as broadly disclosed in the instant specification at Page 7, lines 5-6 and lines 8-9, which is states, ´In a preferred embodiment of the invention, *cells are created and displayed in sets or panels. ... A panel may hold one cell...*' , it is

reasonable to find that the term, "*cells...output panel* " is broad enough to reasonably interprets as "cells table" in Kotler. Thus examiner concludes, reasonably, that the claimed, automatic generation of the required number of destination cells in the output panel is described by Kotler.)

Claim 46,

Kotler and Spencer teach the method of claim 32 and further comprise:

wherein the calculation type is selected from by row, by column, all cells in a panel, selected cells in a panel, accumulated, and non-accumulated.

(In Kotler: Para 139→ Kotler discloses when the user created the formula "=A1" in cell C1. The user selects cell C1, types in an "=" sign to indicate a formula, and then references the cell A1 by moving the mouse pointer 904 to cell A1 and clicking. The spreadsheet objects 106 (namely, CellEditing object 152) capture this reference and pass it to parser 144. The parser 144 recognizes it as a formula creates a formula object and inserts the formula into a cell of cell table 134, as indicated by cell C1. It is reasonable to interprets the steps of said the parser 144 recognizes it as a formula, creates a formula object and inserts the formula into a cell of cell table 134 "without user intervention.

In addition Kotler further teaches the formula manager maintains one or more formula objects 146(1)-146(B) that contain formula information, including the parsed formula expression returned by the parser 144, the current result, the type of formula,

and the current formula state. In addition, there are three types of formulas: normal (e.g., accumulated) semi-calculation, and non-calculation (non-accumulated). The normal formula is reevaluated only when its dependencies change. The semi-calculation formula is reevaluated every time the recalculation engine 142 performs a recalculation operation. The non-calculation formula is never evaluated at all. Non-calculation formulas are a special formula type for handling nested tables (i.e., a table within a table) and nested free floating fields [in Kotler: Para(s) 50-51).)

Claims 24, 26, 27, 38-39, 42-43, 45 and 47 are rejected under 35 U.S.C. 103(a) as being unpatentable over **Kotler** et al., (US 20090083615A1-Continuation of No. 10/961,313, which is a Division of No. 09/599,808 filed 06/21/2000 [hereinafter "Kotler"], in view of **Spencer** et al., (US005603021A - filed 09/02/1994) [hereinafter "Spencer"], and further in view of **Naimat** et al., (US 20050039114A1-Provisional No. 60/487,685 filed 07/16/2003 [hereinafter "Naimat"] .

Claims 24 and 26,

Kotler and Spencer do not expressly teach, but Naimat teaches:

wherein the function component is a Java class that encapsulates the calculation of a specific function; wherein the data component is a Java class that holds an object reference to a specific data source.

(@ Para 56 and 62→ Naimat discloses the web front-end 240 allows an instance user to view and manipulate data in the model using a web browser. In an embodiment, the web front-end 240 is implemented as a JSPs and Java classes and the translator 230 is implemented as a *Java stored procedure* [java. jar file]. After a spreadsheet model is persisted, the persistence component 220 invokes the translator 230. The translator 230 reads and parses the cell formulas, and named range information, and generates a SQL spreadsheet query that represents the formula computation. The generated query is not a complete query, because the full query depends on the values stored for a particular spreadsheet instance. This allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model.)

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler and Spencer's spreadsheet, to include a means of said the function component is a Java class that encapsulates the calculation of a specific function; wherein the data component is a Java class that holds an object reference to a specific data source as taught by Naimat; because they are from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Naimat relates to general

method of modeling spreadsheet formula in Kotler and Spencer. This is done in an iterative manner that enabling the spreadsheet to SQL translator reads the persisted formulas and import/filter data, and translates it to a SQL spreadsheet query. In an embodiment, the translator is implemented as a Java stored procedure. Also, this allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model. The translation from spreadsheet formulas to a SQL spreadsheet query is described in more detail below [In Naimat: @ Para 6].

Claim 27,

Kotler and Spencer do not expressly teach, but Naimat teaches:

further comprising data components including data from an external data source.

(@ Para 104→ Naimat discloses this limitation, as clearly indicated in the cited text [e.g., the modeling tools introduce named references for cells and cell ranges represent data from external tables, a SQL reference spreadsheet is an appropriate representation.])

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler and Spencer's spreadsheet, to include a means of said data components including data from an external data source as taught by Naimat; because they are from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Naimat

relates to general method of modeling spreadsheet formula in Kotler and Spencer. This is done in an iterative manner that enabling the spreadsheet to SQL translator reads the persisted formulas and import/filter data, and translates it to a SQL spreadsheet query. In an embodiment, the modeling tools introduce named references for cells and cell ranges represent data from external tables, a SQL reference spreadsheet is an appropriate representation [In Naimat: @ Para 104].

Claims 38-39,

Kotler and Spencer do not expressly teach, but Naimat teaches:

further including tools for saving a model document as a standalone Java .jar file; and further including tools for saving a model document as a standalone application.

(@ Para 56 and 62→ Naimat discloses the web front-end 240 allows an instance user to view and manipulate data in the model using a web browser. In an embodiment, the web front-end 240 is implemented as a JSPs and Java classes and the translator 230 is implemented as a *Java stored procedure* [java. jar file]. After a spreadsheet model is persisted, the persistence component 220 invokes the translator 230. The translator 230 reads and parses the cell formulas, and named range information, and generates a SQL spreadsheet query that represents the formula computation. The generated query is not a complete query, because the full query depends on the values stored for a particular spreadsheet instance. This allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model.)

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler and Spencer's spreadsheet, to include a means of said the tools for saving a model document as a standalone Java .jar file; and further including tools for saving a model document as a standalone application as taught by Naimat; because they are from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Naimat relates to general method of modeling spreadsheet formula in Kotler and Spencer. This is done in an iterative manner that enabling the spreadsheet to SQL translator reads the persisted formulas and import/filter data, and translates it to a SQL spreadsheet query. In an embodiment, the translator is implemented as a Java stored procedure. Also, this allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model. The translation from spreadsheet formulas to a SQL spreadsheet query is described in more detail below [In Naimat: @ Para 6].

Claims 42 and 43,

Kotler and Spencer do not expressly teach, but Naimat teaches:

further including tools for saving a model document including data;

further including tools for saving a model document without data.

(In Naimat: at the Abstract and @ Para 12 → Naimat discloses the SQL model and its associated data are stored in the database (e.g., saving a model document including

data). The SQL model and its associated data are stored separately in the database (e.g., saving a model document without data).)

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler and Spencer's spreadsheet, to include a means of said including tools for saving a model document including data; further including tools for saving a model document without data as taught by Naimat; because they are from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Naimat relates to general method of modeling spreadsheet formula in Kotler and Spencer. This is done in an iterative manner that enabling the spreadsheet to SQL translator reads the persisted formulas and import/filter data, and translates it to a SQL spreadsheet query. In an embodiment, the translator is implemented as a Java stored procedure. Also, this allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model. The translation from spreadsheet formulas to a SQL spreadsheet query is described in more detail below [In Naimat: @ Para 6]; and allowing the model to be executed on a different set of data. In a further embodiment, a web browser based front-end allows model users to access the SQL model via a web browser, eliminating the need to purchase large numbers of spreadsheet licenses [In Naimat: @ Para 12].

Claim 45,

Kotler and Spencer do not expressly teach, but Naimat teaches:

wherein the data source is selected from a panel of cells and an external database.

(@ Para 64 and 104→ Naimat discloses this limitation, as clearly indicated in the cited text [e.g., the modeling tools introduce named references for cells and cell ranges represent data from external tables, a SQL reference spreadsheet is an appropriate representation.])

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler and Spencer's spreadsheet, to include a means of said the data source is selected from a panel of cells and an external database as taught by Naimat; because they are from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Naimat relates to general method of modeling spreadsheet formula in Kotler and Spencer. This is done in an iterative manner that enabling the spreadsheet to SQL translator reads the persisted formulas and import/filter data, and translates it to a SQL spreadsheet query. In an embodiment, the modeling tools introduce named references for cells and cell ranges represent data from external tables, a SQL reference spreadsheet is an appropriate representation [In Naimat: @ Para 104].

Claim 47,

Kotler and Spencer do not expressly teach, but Naimat teaches:

wherein the standalone application is selected from a Java Applet, a serverside application, and an .exe-file.

(@ Para 56 and 62→ Naimat discloses the web front-end 240 allows an instance user to view and manipulate data in the model using a web browser. In an embodiment, the web front-end 240 is implemented as a JSPs and Java classes. It is reasonably to interpret the web front end includes a Java Applet, since the web front-end 240 allows an instance user to view and manipulate data in the model using a web browser; web browser is implemented as a JSPs and Java classes.)

Accordingly, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Kotler and Spencer's spreadsheet, to include a means of said the standalone application is selected from a Java Applet, a serverside application, and an .exe-file as taught by Naimat; because they are from the analogous art of composing spreadsheet formula model in a spreadsheet engine. Therefore, the artisan would have well appreciated that Naimat relates to general method of modeling spreadsheet formula in Kotler and Spencer. This is done in an iterative manner that enabling the spreadsheet to SQL translator reads the persisted formulas and import/filter data, and translates it to a SQL spreadsheet query. In an embodiment, the translator is implemented as a Java stored procedure. Also, this allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model. The translation from spreadsheet formulas to a SQL spreadsheet query is described in more detail below [In Naimat: @ Para 6].

It is noted that any citations to specific, pages, columns, lines, or figures in the prior art references and any interpretation of the references should not be considered to be limiting in any way. A reference is relevant for all it contains and may be relied upon for all that it would have reasonably suggested to one having ordinary skill in the art. See, MPEP 2123.

Response to Arguments

Applicant's arguments filed with current paper have been considered but are moot in view of the new ground(s) of rejections as cited above.

It is noted; Examiner introduces Kotler patent to address some of the amended limitation (see above rejections for details).

In addition, Examiner retains references Naimat, and Spencer, since; ***Spencer*** discloses a Formula Composer 136 is a visual tool for creating, editing, and debugging spreadsheet formulas. Formula Composer Dialog providing multiple views of a formula, including: (1) a Formula Outline Pane to examine the structure of a formula, edit parts of the formula, and trace cell references and block names; (2) a Subexpression Edit Field for text editing of a formula or portion thereof; and (3) a Function Panel to facilitate input of *@-functions and their arguments*. With these different views, the user can easily

choose the right @-functions and enter all necessary information correctly (At Figure 3A and at the Abstract and Col. 11, Lines 10-15). Also Spencer further teaches The "@" button 303, which appears to the left of the Subexpression Field, displays Functions Dialog 321, as shown in FIG. 3B. The Dialog 321 includes a Function Category List 323 and a Function (name) list 325. For example, that a user has selected the "Amortized Accumulated Interest" (AMAINT) function from the list 325. In response, the system updates the Dialog 321, as shown by the Dialog 321a in FIG. 3C. The AMAINT function is shown selected, at 325a. The information panel has been updated accordingly, shown at 327a. Upon the user selecting the OK button 329, the AMAINT @-function will be placed in the Subexpression Edit Field, as shown by the Dialog 301a of FIG. 3D. The user may now proceed to further change the formula using the Formula Outline Pane (In Spencer: @ Col. 12 Lines 10-35 and illustrates in Fig(s) 3(a), (b), (c) and (d)).
In addition, since; ***Naimat*** discloses the web front-end; that allows an instance user to view and manipulate data in the model using a web browser. In an embodiment, the web front-end 240 is implemented as a JSPs and Java classes and the translator 230 is implemented as a *Java stored procedure* [java. jar file]. After a spreadsheet model is persisted, the persistence component 220 invokes the translator 230. The translator 230 reads and parses the cell formulas, and named range information, and generates a SQL spreadsheet query that represents the formula computation. The generated query is not a complete query, because the full query depends on the values stored for a particular spreadsheet instance. This allows for reusability of the generated SQL to prevent reparsing when another user requests an instance of the model (@ Para 56 and 62).

Also @ Para 64 and 104→ Naimat discloses the modeling tools introduce named references for cells and cell ranges represent data from external tables, a SQL reference spreadsheet is an appropriate representation.])

Accordingly, for at least all the above evidence, therefore the Examiner respectfully maintains the rejection of claims 21-47 at this time.

Conclusion

Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly **THIS ACTION IS MADE FINAL** See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a). Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action.

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Quoc A. Tran whose telephone number is 571-272-8664. The examiner can normally be reached on Mon through Fri 8AM - 5PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Doug Hutton can be reached on (571)272-4137. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Quoc A, Tran/
Patent Examiner

/DOUG HUTTON/
Supervisory Patent Examiner, Art Unit 2176